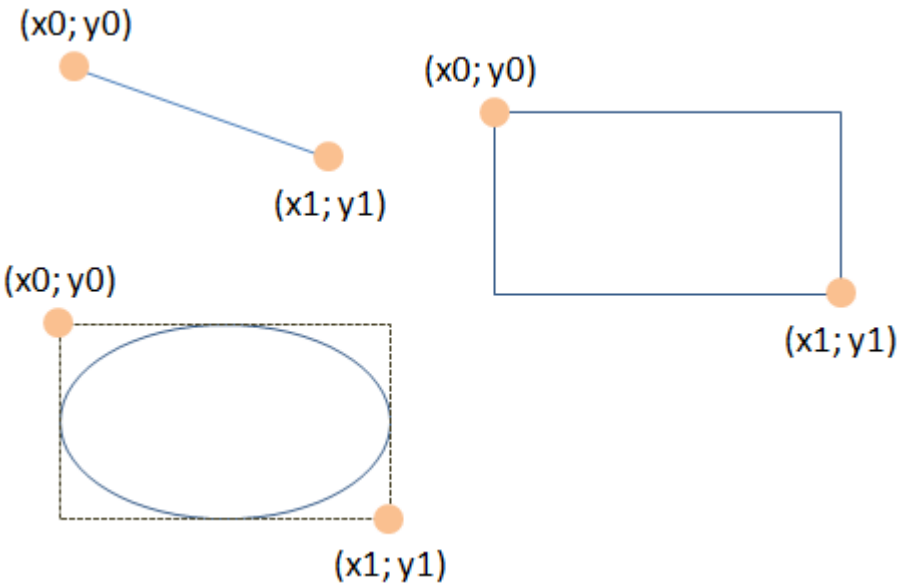
Итак, мы с вами подошли, наверное, к самому главному моменту в ООП – **наследованию** классов. **Наследование – это то, что превращает программу из простого набора объектов в настоящую концепцию программирования, то, что выводит нас на совершенно новый уровень проектирования и написания программного кода**. И совсем скоро, вы своими глазами сможете в этом убедиться.

Для объяснения работы механизма наследования я возьму классический пример работы с геометрическими фигурами на плоскости. Будем полагать, что их можно описать четырьмя числами: координатами левого верхнего угла и правого нижнего углов:



Тогда, например, класс для прямых линий (Line) можно объявить следующим образом:

```cpp
class Line {
    int x0{0}, y0{0}, x1{0}, y1{0};
    double length{0.0};

public:
    void set_coords(int x0, int y0, int x1, int y1)
    {
        this->x0 = x0; this->y0 = y0;
        this->x1 = x1; this->y1 = y1;
    }
};
```

Здесь дополнительно прописана переменная length, содержащая длину линии, и эта переменная характерна только для класса Line.

По аналогии можно было бы объявить и остальные классы, но я пока не стану этого делать. Мы отметим следующее. Все фигуры имеют общие элементы в описании – это координаты x0, y0, x1, y1. Было бы логично выделить их в общий класс. Давайте это сделаем. Определим класс для произвольных геометрических фигур с именем GeomBase:

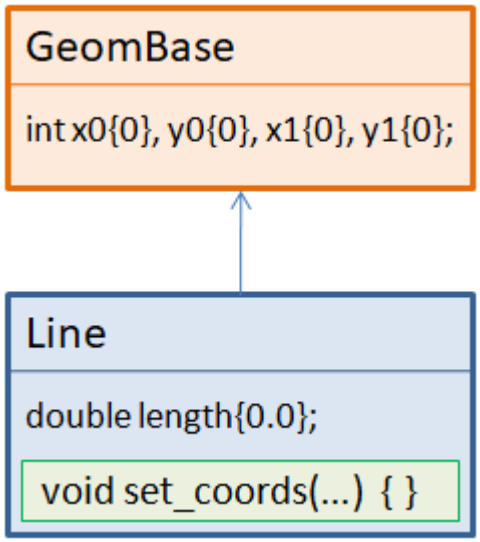
```language-cpp
class GeomBase {
private:
    int x0{0}, y0{0}, x1{0}, y1{0};
};
```

И класс Line унаследуем от этого класса:

```language-cpp
class Line : public GeomBase {
private:
    double length{0.0};

public:
    void set_coords(int x0, int y0, int x1, int y1)
    {
        this->x0 = x0; this->y0 = y0;
        this->x1 = x1; this->y1 = y1;
    }
};
```

Обратите внимание, как записывается наследование классов в языке C++. Ставится двоеточие, затем, режим наследование (в примере public) и имя базового класса GeomBase. О режимах наследования речь еще впереди.



Соответственно, класс, от которого выполняется наследование, называют **базовым, родительским**, а класс наследник – **дочерним, производным**.

Однако при такой организации у нас возникли некоторые проблемы. В методе set_coords стали недоступны переменные x0, y0, x1, y1. С чем это связано? Смотрите, эти **переменные расположены в секции private базового класса GeomBase. А эта секция ограничивает доступ к ее элементам в пределах текущего класса, и не важно, что класс Line является дочерним для класса GeomBase**. Мы можем работать с координатами только в классе GeomBase. Например, если перенести метод set_coords в базовый класс, то никаких ошибок не будет:

```language-cpp
class GeomBase {
private:
    int x0{0}, y0{0}, x1{0}, y1{0};
public:
    void set_coords(int x0, int y0, int x1, int y1)
    {
        this->x0 = x0; this->y0 = y0;
        this->x1 = x1; this->y1 = y1;
    }
};
```

При этом в дочернем классе Line совершенно спокойно можно вызывать публичный метод set_coords и обращаться к любым другим публичным полям базового класса. Давайте для примера добавим в базовый класс еще один метод для получения текущих координат:

```language-cpp
    void get_coords(int& x0, int& y0, int& x1, int& y1)
    {
        x0 = this->x0; y0 = this->y0;
        x1 = this->x1; y1 = this->y1;
    }
```

А в дочернем классе определим метод draw для отображения объекта линии:

```language-cpp
class Line : public GeomBase {
private:
    double length{0.0};
public:
    void draw()
    {
        int x0, y0, x1, y1;
        get_coords(x0, y0, x1, y1);
```

```
        printf("Line: %d, %d, %d, %d\n", x0, y0, x1, y1);
    }
};
```

После этого в функции main можно создать объект класса Line и вызвать для него соответствующие методы:

```language-cpp
int main()
{
    Line ln;

    ln.set_coords(1, 2, 10, 20);
    ln.draw();

    return 0;
}
```

Режим доступа protected

Однако обращение к переменным-координатам базового класса GeomBase в методе draw дочернего класса Line через публичные методы – не лучшее решение. Было бы хорошо эти координаты сделать закрытыми от доступа извне, но сделать доступными во всех производных от него классах. И такой режим в ООП существует. Он называется **protected**. Если мы в базовом классе вместо режима private укажем protected:

```language-cpp
class GeomBase {
protected:
    int x0{0}, y0{0}, x1{0}, y1{0};
public:
...
};
```

то к переменным x0, y0, x1, y1 можно обращаться напрямую из всех его дочерних классов. Соответственно, метод draw класса Line значительно упрощается до одной строчки:

```language-cpp
class Line : public GeomBase {
private:
    double length{0.0};
public:
    void draw()
    {
        printf("Line: %d, %d, %d, %d\n", x0, y0, x1, y1);
```

```
    }
};
```

Видите, как это бывает удобно. При этом вне классов получить доступ к этим переменным по-прежнему запрещено. Следующая команда приведет к ошибке:

```language-cpp
Line ln;
ln.x0 = 10; // ошибка
```

То есть, в классах можно прописывать три разных режима:

- private – доступ в пределах текущего класса;
- protected – доступ в пределах текущего и всех дочерних от него классах;
- public – общий доступ (внутри и за пределами класса).

Надо отметить, что помещать переменные в секцию protected следует с особой осторожностью, т.к. дочерние классы получают к ним прямой доступ с произвольной логикой их обработки. Поэтому protected-переменные следует воспринимать, как особый вид публичных переменных и при проектировании логики класса (где они объявлены) относиться к ним как к публичным.